Available online at www.sciencedirect.com

ScienceDirect

Journal of Computer and System Sciences 74 (2008) 546–556

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

www.elsevier.com/locate/jcss

Learning a hidden graph using $O(\log n)$ queries per edge

Dana Angluin [a,*], Jiang Chen [b]

[a] *Department of Computer Science, Yale University, USA*
[b] *Center for Computational Learning Systems, Columbia University, USA*

Received 4 November 2004; received in revised form 17 March 2005

Available online 12 June 2007

Abstract

We consider the problem of learning a general graph using edge-detecting queries. In this model, the learner may query whether a set of vertices induces an edge of the hidden graph. This model has been studied for particular classes of graphs by Grebinski and Kucherov [V. Grebinski, G. Kucherov, Optimal query bounds for reconstructing a Hamiltonian cycle in complete graphs, in: Fifth Israel Symposium on the Theory of Computing Systems, 1997, pp. 166–173] and Alon et al. [N. Alon, R. Beigel, S. Kasif, S. Rudich, B. Sudakov, Learning a hidden matching, in: The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002, pp. 197–206], motivated by problems arising in genome sequencing. We give an adaptive deterministic algorithm that learns a general graph with n vertices and m edges using $O(m \log n)$ queries, which is tight up to a constant factor for classes of non-dense graphs. Allowing randomness, we give a 5-round Las Vegas algorithm using $O(m \log n + \sqrt{m} \log^2 n)$ queries in expectation. We give a lower bound of $\Omega((2m/r)^{r/2})$ for learning the class of non-uniform hypergraphs of dimension r with m edges.

Keywords: Query learning; Edge-detecting queries; Graphs; Hypergraphs; Monotone DNF formulas; Chemical reaction networks

1. Introduction

The problem of learning a hidden graph is the following. Imagine that there is a graph $G = (V, E)$ whose vertices are known to us and whose edges are not. We wish to determine all the edges of G by making *edge-detecting queries* of the following form

$Q_G(S)$: does S include at least one edge of G?

where $S \subseteq V$. The query $Q_G(S)$ is answered 1 or 0, indicating whether S contains both ends of at least one edge of G or not. We abbreviate $Q_G(S)$ to $Q(S)$ whenever the choice of G is clear from the context. The edges and non-edges of G are completely determined by the answers to $Q(\{u, v\})$ for all unordered pairs of vertices u and v; however, we seek algorithms that use significantly fewer queries when G is not dense.

* Corresponding author.
E-mail addresses: angluin@cs.yale.edu (D. Angluin), criver@cs.columbia.edu (J. Chen).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2007.06.006

This type of query may be motivated by the following scenario. We are given a set of chemicals, some pairs of which react and others do not. When multiple chemicals are combined in one test tube, a reaction is detectable if and only if at least one pair of the chemicals in the tube react. The task is to identify which pairs react using as few experiments as possible. The time needed to compute which experiments to do is a secondary consideration, though it is polynomial for the algorithms we present.

An important aspect of an algorithm in this model is its adaptiveness. An algorithm is *non-adaptive* if the whole set of queries it makes is chosen before the answers to any queries are known. An algorithm is *adaptive* if the choice of later queries may depend on the answers to earlier queries. Although adaptiveness is powerful, non-adaptiveness is desirable in practice to permit the queries (or experiments) to be parallelized. A *multiple-round algorithm* consists of a sequence of rounds in which the set of queries made in a given round may depend only on the answers to queries asked in preceding rounds. Since the queries in each round may be parallelized, it is desirable to keep the number of rounds small. A non-adaptive algorithm is a 1-round algorithm.

Another important aspect of an algorithm is what assumptions may be made about the graph G; this is modeled by assuming that G is drawn from a known class of graphs. Previous work has mainly concentrated on identifying a graph G drawn from the class of graphs isomorphic to a fixed known graph. The cases of Hamiltonian cycles and matchings have specific applications to genome sequencing, which are explained in the papers cited below. Grebinski and Kucherov [7] give a deterministic adaptive algorithm for learning Hamiltonian cycles using $O(n \log n)$ queries. Beigel et al. [6] describe an 8-round deterministic algorithm for learning matchings using $O(n \log n)$ queries, which has direct application in genome sequencing projects. Alon et al. [3] give a 1-round Monte Carlo algorithm for learning matchings using $O(n \log n)$ queries, which succeeds with high probability. On the other hand, they show a lower bound of $\Omega(\binom{n}{2})$ for learning matchings with a deterministic 1-round algorithm. They also give a nearly matching upper bound in this setting. Alon and Asodi [2] give bounds for learning stars and cliques with a deterministic 1-round algorithm. Considerable effort has been devoted to optimizing the implied constants in these results.

In this paper, we are interested in the power of edge-detecting queries from a more theoretical point of view. In particular, we consider the problem of learning more general classes of graphs. Because of this focus, in this paper, we are more interested in asymptotic results than optimizing constants.

Let n denote the number of vertices and m the number of edges of G. Clearly n is known to the algorithm (since V is known), but m may not be. In Section 3, we give a deterministic adaptive algorithm to learn any graph using $O(m \log n)$ queries. The algorithm works without assuming m is known. For Hamiltonian cycles, matchings, and stars, our algorithm uses $O(n \log n)$ queries. In Section 4, we give a 1-round Monte Carlo algorithm for all graphs of degree at most d using $O(dn(\log n + \log 1/\delta))$ queries that succeeds with probability at least $1 - \delta$ assuming d is known. Note that Hamiltonian cycles and matchings are both degree bounded by constants. This algorithm takes $O(n \log n)$ queries in both cases. In Section 5, we consider constant-round algorithms for general non-dense graphs. We first briefly describe a 4-round Las Vegas algorithm using $O(m \log n + \sqrt{m} \log^2 n)$ queries in expectation, assuming m is known. If m is not known, we give a 5-round Las Vegas algorithm that uses as few queries. Note that $O(\sqrt{m} \log^2 n)$ is negligible when $m = \Omega(\log^2 n)$. Therefore, the 5-round algorithm achieves $O(\log n)$ queries per edge unless the graph is very sparse, i.e. $m = o(\log^2 n)$.

In Section 6 we consider the problem of learning hypergraphs. The information-theoretic lower bound implies that $\Omega(rm \log n)$ queries are necessary for learning the class of hypergraphs of dimension r with m edges. We show further that no algorithm can learn this class of hypergraphs using $o((2m/r)^{r/2})$ queries. However, non-uniformity of hypergraphs does play an important role in our construction of lower bound. Angluin and Jiang [5] give an algorithm that learns an r-uniform hypergraph with m edges using $O(poly(2^r, \log n) \cdot m)$ edges with high probability.

The graph learning problem may also be viewed as the problem of learning a monotone disjunctive normal form (DNF) boolean formula with terms of size 2 using membership queries only. Each vertex of G is represented by a variable and each edge by a term containing the two variables associated with the endpoints of the edge. A membership query assigns 1 or 0 to each variable, and is answered 1 if the assignment satisfies at least one term, and 0 otherwise, that is, if the set of vertices corresponding to the variables assigned 1 contains both endpoints of at least one edge of G. Similarly, a hyperedge with r vertices corresponds to a term with r variables. Thus, our results apply also to learning the corresponding classes of monotone DNF formulas using membership queries. The graph-theoretic formulation provides useful intuitions.

2. Preliminaries

A hypergraph is a pair $H = (V, E)$ such that E is a subset of the power set of V, where V is the set of vertices and E is the set of edges. A set $S \subseteq V$ is an *independent set* of G if it contains no edge of H. The *degree* of a vertex is the number of edges of H that contain it. If S is a set of vertices, then the *neighbors* of S are all those vertices v not in S such that $\{u, v\}$ is contained in an edge of H for some $u \in S$. We denote the set of neighbors of S by $\Gamma(S)$. The dimension of a hypergraph H is the cardinality of the largest set in E. H is said to be r-uniform if E contains only sets of size r. In an r-uniform hypergraph, a set of vertices of size r is called a *non-edge* if it is not an edge of H.

A undirected simple graph G with no self loops is a just 2-uniform hypergraph. Thus the edges of $G = (V, E)$ may be considered to be a subset of the set of all unordered pairs of vertices of G. A c-coloring of a graph G is a function from V to $\{1, 2, \ldots, c\}$ such that no edge of G has both endpoints mapped to the same color. The set of vertices assigned the same color by a coloring is a *color class* of the coloring.

A graph is d-regular if all its nodes have degree d. A Hamiltonian cycle is a cycle in a graph that visits each node exactly once. A matching in a graph is a set of edges that do not share vertices. A star graph is a tree with a central node to which every other node is connected with an edge.

Here are three inequalities that we use.

Proposition 1. *If* $0 \leqslant x \leqslant 1$,

$$1 - x \leqslant e^{-x}.$$

Proposition 2. *If* $0 < x \leqslant 1$,

$$(1-x)^{\frac{1}{x}} \geqslant \frac{2(1-x)}{e(2-x)} \geqslant \frac{(1-x)}{e}.$$

Proposition 3. *(See Hoeffding's inequality [8].) Let* x_i $(i = 1, \ldots, n)$ *be* n *independent random variables bounded in* $[0, 1]$. *Let* $S = \sum_{i=1}^{n} x_i$. *Then for any* $\epsilon > 0$,

$$\Pr\big[S - E[S] \geqslant \epsilon\big] \leqslant e^{-2\epsilon^2/n}.$$

3. An adaptive algorithm

The main result of this section is the following.

Theorem 3.1. *There is a deterministic adaptive algorithm that identifies any graph* G *drawn from the class of all graphs with* n *vertices using* $O(m \log n)$ *edge-detecting queries, where* m *is the number of edges of* G.

By a counting argument, this upper bound is tight up to a constant factor for certain classes of non-dense graphs.

Theorem 3.2. *For any* $0 < \epsilon < 2$, $\Omega(\epsilon m \log n)$ *edge-detecting queries are required to identify a graph* G *drawn from the class of all graphs with* n *vertices and* $m = n^{2-\epsilon}$ *edges.*

Proof. There are

$$\binom{\binom{n}{2}}{m} \geqslant \left(\frac{\binom{n}{2}}{m}\right)^m$$

graphs that have m edges. Because each query has a 1-bit answer, the information-theoretic lower bound implies at least $m \log(\binom{n}{2}/m) = \Omega(\epsilon m \log n)$ queries in the worst case. □

We begin by presenting a simple adaptive algorithm for the case of finding the edges between two known independent sets of vertices in G using $O(\log n)$ queries per edge. Assume that S_1 and S_2 are two known, non-empty

independent sets of vertices in G. Also assume that $|S_1| \leqslant |S_2|$ and there are s edges between S_1 and S_2, where $s > 0$. The following algorithm works without prior knowledge of s.

Algorithm 1

Assuming $|S_1| \leqslant |S_2|$.

1. If both S_1 and S_2 are singleton sets, there is exactly one edge between S_1 and S_2. Mark this edge and return.
2. In this case, $|S_2| > 1$.
   (a) Divide S_2 arbitrarily into two sets S_{21} and S_{22}, such that $|S_{21}| = \lfloor |S_2|/2 \rfloor$ and $|S_{22}| = \lceil |S_2|/2 \rceil$.
   (b) Make queries on $S_1 \cup S_{21}$ and $S_1 \cup S_{22}$.
   (c) For $j = 1, 2$, solve the problem recursively for S_1 and S_{2j} if the query on $S_1 \cup S_{2j}$ is answered 1.

Lemma 3.3. *Algorithm* 1 *identifies edges between S_1 and S_2 using no more than* $4s(\log |S_2| + 1)$ *edge-detecting queries.*

Proof. If we consider the computation tree for this algorithm, the maximum depth does not exceed $\lceil \log |S_2| \rceil + \lceil \log |S_1| \rceil \leqslant 2(\log |S_2| + 1)$ and there are at most s leaves in the tree (corresponding to the s edges of G that are found.) At each internal node of the computation tree, the algorithm asks at most 2 queries. Therefore, the algorithm asks at most $4s(\log |S_2| + 1)$ queries. □

If S_1 and S_2 are not independent sets in G, the problem is more complex because we must eliminate interference from the edges of G induced by S_1 or S_2. If we happen to know the edges of G induced by S_1 and S_2, and we color the two induced graphs, then each color class is an independent set in G. Then the edges between a color class in S_1 and a color class in S_2 can be identified using Algorithm 1. Because every edge between S_1 and S_2 belongs to one such pair, it suffices to consider all such pairs. The next lemma formalizes this idea.

Lemma 3.4. *For $i = 1, 2$ assume that S_i is a set of vertices that includes s_i edges of G, where s_1 and s_2 are not both* 0*, and assume that these edges are known. Also assume that $|S_1| \leqslant |S_2|$ and there are $s > 0$ edges between S_1 and S_2. Then these edges can be identified adaptively using no more than* $4(s \log |S_2| + s + s_1 + s_2)$ *edge-detecting queries.*

We observe the following fact about vertex coloring.

Fact 1. A graph with m edges can be $\lfloor \sqrt{2m} + 1 \rfloor$-colored. Furthermore, the coloring can be constructed in polynomial time.

To see this, we successively collapse pairs of vertices not joined by an edge until we obtain the complete graph on t vertices, which can be t-colored and has $t(t-1)/2 \leqslant m$ edges. This yields a t-coloring of the original graph because no edge joins vertices that are collapsed into the same final vertex.

Proof of Lemma 3.4. Using the preceding Fact 1, for $i = 1, 2$, we may color the subgraph of G induced by S_i using at most $\lfloor \sqrt{2s_i} + 1 \rfloor$ colors. Each color class is an independent set in G. The edges between S_1 and S_2 can be divided into the sets of edges between pairs of color classes from S_1 and S_2. For each pair of color classes, one from S_1 and one from S_2, we query the union of the two classes to determine whether there is any edge of G between the two classes. If so, then using the algorithm in Lemma 3.3, we can identify the edges between the two classes with no more than $4(\log |S_2| + 1)$ queries per edge. To query the union of each pair of color classes requires at most $(\lfloor \sqrt{2s_1} + 1 \rfloor)(\lfloor \sqrt{2s_2} + 1 \rfloor)$ queries, which does not exceed $(1 + \sqrt{2})(s_1 + s_2) + 1$. Thus, in total, we use no more than $4(s \log |S_2| + s + s_1 + s_2)$ edge-detecting queries. □

Now we are able to present our adaptive algorithm to learn a general graph $G = (V, E)$ with $O(\log n)$ queries per edge. One query with the set V suffices to determine whether E is empty, so we assume that $|E| > 0$.

Algorithm 2 (*Adaptive algorithm*)

1. If $|V| = 2$, mark the pair of vertices in V as an edge and return.
2. Divide V into halves S_1 and S_2. Ask $Q(S_1)$ and $Q(S_2)$.
3. Recursively solve the problem for S_i if $Q(S_i) = 1$, for $i = 1, 2$.
4. Use the algorithm in Lemma 3.4 to identify the edges between S_1 and S_2.

Proof of Theorem 3.1. We give an inductive proof that Algorithm 2 uses no more than $12m \log n$ edge-detecting queries to learn a graph G with n vertices and $m > 0$ edges. This clearly holds when $n = 2$. Assume that for some $n \geqslant 3$, every graph with $n' < n$ vertices and $m' > 0$ edges is learnable with at most $12m' \log n'$ edge-detecting queries. Assume S_i includes s_i edges of G, for $i = 1, 2$. Since $|S_2| \geqslant |S_1|$, the number of queries required to learn G is at most

$$\big(12(s_1 + s_2) + 4(m - s_1 - s_2)\big) \log|S_2| + 4m + 2$$

using the inductive hypothesis and Lemma 3.4.

We know that $\log|S_2| \leqslant \log((n+1)/2) \leqslant \log n - 1/2$, when $n \geqslant 3$. Then for $n \geqslant 3$, the above expression is at most $12m \log n$ because $m \geqslant 1$. This concludes the induction. □

This shows that any graph is adaptively learnable using $O(\log n)$ queries per edge. This algorithm can be parallelized into $O(\log^2 n)$ non-adaptive rounds, because Algorithm 1 can be parallelized to find all edges between two independent sets in $O(\log n)$ rounds (queries on $S_1 \cup S_{21}$ and $S_1 \cup S_{22}$ can be made in parallel and queries in different recursive calls can be made in parallel), and the depth of recursion of Algorithm 2 is $O(\log n)$. In subsequent sections we develop randomized algorithms that achieve a constant number of rounds.

4. Bounded degree graphs

In this section, we present a randomized non-adaptive algorithm to learn any graph with bounded degree d, where we assume that $d = o(n)$ and d is known to the algorithm. The algorithm uses $O(dn(\log n + \log 1/\delta))$ queries and succeeds with probability at least $1 - \delta$. Our algorithm is a generalization of that of Alon et al. [3] to learn a hidden matching using $O(n \log n)$ queries. In contrast to their results, we use sampling with replacement and do not attempt to optimize the constants, as our effort is to map out what is possible in the general case.

The key observation is that every pair of vertices in S is discovered to be a non-edge of G if $Q(S) = 0$. The algorithm asks a set of queries with random sets of vertices with the goal of discovering all of the non-edges of G.

For a probability p, a *p-random* set P is obtained by including each vertex independently with probability p. Each query is an independently chosen p-random set. After all the queries are answered, those pairs of vertices that have not been discovered to be non-edges are output as edges in G. The algorithm may fail by not discovering some non-edge of G, and we bound the probability of failure by δ for an appropriate choice of p and number of queries.

For a given non-edge $\{u, v\}$ in G, the probability that both u and v are included in a p-random set P is p^2. Given that u and v are included in P, the probability that P has no edge of G is bounded below using the following lemma. Let $N_G(p)$ denote the probability that a p-random set includes no edge of G.

Lemma 4.1. *Suppose I is an independent set in G, and $\Gamma(I)$ is the set of neighbors of vertices in I. Suppose P is a p-random set.* $\Pr\{Q(P) = 0 \mid I \subseteq P\}$ *is at least*

$$(1 - p)^{|\Gamma(I)|} \cdot N_G(p).$$

Proof. Let G' be the induced subgraph of G on $V - I - \Gamma(I)$. It is easy to verify that $N_{G'}(p) \geqslant N_G(p)$. Independence in the selection of the vertices in P implies that $\Pr\{Q(P) = 0 \mid I \subseteq P\}$ is the product of the probability that P contains no vertices in $\Gamma(I)$, which is $(1-p)^{|\Gamma(I)|}$, and the probability that given the previous event P has no edge of G, which is $N_{G'}(p)$. □

By the union bound, we know that $N_G(p) \geqslant 1 - mp^2$. Also, $\Gamma(\{u, v\}) \leqslant 2d$ because the degree of each vertex of G is bounded by d. Therefore,

$$\Pr\{Q(P) = 0 \mid u, v \in P\} \geqslant (1-p)^{2d}(1-mp^2) \geqslant 1 - 2dp - mp^2.$$

Since d is assumed to be known to the algorithm, we choose $p = 1/\sqrt{dn}$. Then the above expression is at least $1 - 2\sqrt{d/n} - m/dn \geqslant 1/2 - o(1)$. (Recall that we assume $d = o(n)$.) Therefore, the probability $\{u, v\}$ is shown to be a non-edge of G by one random query is at least

$$p^2 \cdot \left(\frac{1}{2} - o(1)\right) = \frac{1}{2dn} \cdot \big(1 - o(1)\big).$$

The probability that a non-edge $\{u, v\}$ is not discovered to be a non-edge using $2dn \cdot (1 + o(1)) \cdot (2\ln n + \ln 1/\delta)$ queries is at most δ/n^2 (using Proposition 1). Thus, the probability that some non-edge of G is not discovered after this many queries is bounded by δ.

Therefore, we have proved the following.

Theorem 4.2. *For any $0 < \delta < 1$, there is a Monte Carlo non-adaptive algorithm that identifies any graph G drawn from the class of graphs with bounded degree d with probability at least $1 - \delta$ using $O(dn(\log n + \log 1/\delta))$ edge-detecting queries, where n is the number of vertices.*

When $\delta = 1/poly(n)$, for d-regular graphs, this algorithm uses $O(m \log n)$ queries. In particular, for matchings and Hamiltonian cycles, the algorithm uses $O(n \log n)$ queries.

5. Constant-round algorithms

The algorithm in the previous section is not query-efficient when G is far from regular, e.g. we get a bound of $O(n^2 \log n)$ to learn a star with only $n - 1$ total edges, because the maximum degree is large. To obtain a query-efficient algorithm for a more general class of graphs, we consider constant-round algorithms, in which the set of queries in a given round may depend on the answers to queries in preceding rounds. For each round of the algorithm, a *pseudo-edge* is any pair of vertices that has not been discovered to be a non-edge of G in any preceding round; this includes all the edges of G and all the (as yet) undiscovered non-edges of G.

In a multiple-round algorithm, there is the option of a final *cleanup* round, in which we ask a query for each remaining pseudo-edge, yielding a Las Vegas algorithm instead of a Monte Carlo algorithm. For example, if we add a cleanup round to the algorithm in the previous section, we can get a 2-round Las Vegas algorithm that always answers correctly and uses $O(dn \log n)$ queries in expectation.

The algorithm in the previous section assumes d is known. In this section, we first sketch the intuitions of a 4-round Las Vegas algorithm that learns a general graph using an expected $O(m \log n + \sqrt{m} \log^2 n)$ queries, assuming m is known.

We then develop a 5-round Las Vegas algorithm that learns a general graph using as many queries without assuming m is known.

Each vertex of G is classified as a *low-degree vertex*, if its degree does not exceed $\sqrt{m}$, or a *high-degree vertex* otherwise. A non-edge of G is a *low-degree non-edge* if both vertices in the pair are low-degree vertices.

For the first round we choose the sample probability $p = 1/\sqrt{2m}$. (Recall that we are assuming m is known in this sketch.) Using Lemma 4.1, the probability that a particular low-degree non-edge of G is shown to be a non-edge by a query with a p-random set is at least

$$p^2 \cdot (1-p)^{2\sqrt{m}}\big(1 - m \cdot p^2\big)$$

which is $\Omega(1/m)$. Thus, $O(m \log n)$ queries with p-random sets suffice to identify all the low-degree non-edges of G in the first round with probability at least $1 - n^{-2}$.

Because the number of high-degree vertices is at most $2\sqrt{m}$, we can afford to query all pairs of them in the cleanup round. We therefore concentrate on non-edges containing one high-degree and one low-degree vertex. To discover these non-edges, we need a smaller sampling probability ($p = o(1/\sqrt{m})$), but choosing a sample probability that is too small runs the risk of requiring too many queries.

The right choice of a sampling probability p differs with the degree of each individual high-degree vertex, so in the second round we estimate such p's. In the third round, we use the estimated p's to identify non-edges containing

a high-degree and a low-degree vertex. In the cleanup round we ask queries on every remaining pseudo-edge. In fact, since the actual degrees of the vertices are not known, the sets of high-degree and low-degree vertices must be approximated.

The above paragraphs sketch the intuitions for a 4-round algorithm when m is known. If m is unknown, one plausible idea would be to try to estimate m sufficiently accurately by random sampling in the first round, and then proceed with the algorithm sketched above. This idea does not seem to work, but analyzing it motivates the development of our final 5-round algorithm.

First we have the following "obvious" lemma: as we increase the sampling probability p, we are more likely to include an edge of G in a p-random set.

Lemma 5.1. *Assuming $m > 0$, $N_G(p)$ is strictly decreasing as p increases.*

Proof. Let I_k be the number of independent sets of size k in G and we assume $I_0 = 1$.

$$N_G(p) = \sum_{k=0}^{n} I_k p^k (1-p)^{n-k}. \tag{1}$$

The derivative of $N_G(p)$ with respect to p is

$$\begin{aligned}\big(N_G(p)\big)'_p &= \sum_{k=0}^{n} I_k \big(k p^{k-1}(1-p)^{n-k} - (n-k)p^k(1-p)^{n-k-1}\big) \\ &= -\sum_{k=0}^{n} \big(I_k(n-k) - I_{k+1}(k+1)\big) p^k (1-p)^{n-k-1}. \end{aligned} \tag{2}$$

We will show that $\forall k$, $I_k(n-k) \geqslant I_{k+1}(k+1)$. Let us count I_{k+1}. Each independent set of size $k+1$ can be obtained by adding one element to an independent set of size k. Therefore the total number is at most $I_k(n-k)$. Each independent set of size $k+1$ is counted exactly $k+1$ times in this counting process. Therefore we have $I_k(n-k) \geqslant I_{k+1}(k+1)$. When $m > 0$, $\forall k > 0$, the above inequality is strict. Therefore, $(N_G(p))'_p$ is negative. □

It follows that since $N_G(0) = 1$ and $N_G(1) = 0$, there exists a unique $p_*(G)$ such that $N(p_*(G)) = 1/2$. In other words, $p_*(G)$ is the sampling probability p that makes an edge-detecting query with a p-random set equally likely to return 0 or 1, maximizing the information content of such queries.

It is plausible to think that $p_*(G)$ will reveal much about m. However, $p_*(G)$ also depends strongly on the topology of G. Consider the following two graphs: G_M, a matching with m edges, and G_S, a star with m edges. We have

$$\begin{aligned} N_{G_M}(p) &= \big(1-p^2\big)^m, \\ N_{G_S}(p) &= 1 - p + p(1-p)^m. \end{aligned}$$

Therefore, we have $p_*(G_M) = O(1/\sqrt{m})$ but $p_*(G_S) > 1/2$. We believe that such a gap in $p_*(G)$'s of two different topologies lies behind the difficulty of estimating m in one round.

Although our effort to estimate m has been thwarted, $p_*(G)$ turns out to be the sampling probability that will help us identify most of the non-edges in the graph. We will use $N(p)$ instead of $N_G(p)$ and p_* instead of $p_*(G)$ when the choice of G is clear from the context.

First, we have rough upper and lower bounds for p_*.

$$\frac{1}{\sqrt{2m}} \leqslant p_* \leqslant \frac{\sqrt{2}}{2},$$

observing that $1 - mp^2 \leqslant N(p) \leqslant 1 - p^2$. The fact that p_* helps us identify most of the non-edges is made clear in the following two lemmas.

Lemma 5.2. *Let $\{u, v\}$ be a non-edge of G in which the degrees of u and v do not exceed $2/p_*$. Then a query on a p_*-random set identifies $\{u, v\}$ as a non-edge with probability at least $\Omega(1/m)$.*

Proof. According to Lemma 4.1, the probability that the non-edge $\{u, v\}$ is identified by a query on a p_*-random set is at least

$$p_*^2(1-p_*)^{4/p_*} \cdot N(p_*).$$

We know that $p_* \leqslant \sqrt{2}/2$. According to Proposition 2, $(1-p_*)^{4/p_*} = \Omega(1)$. Combining this with the facts that $p_* \geqslant 1/\sqrt{2m}$ and $N(p_*) = 1/2$, we have that the probability is $\Omega(1/m)$. □

Examining the proof of Lemma 5.2 we can see that rather than requiring the sampling probability p be exactly p_*, it is sufficient to require upper and lower bounds as follows: $p = \Omega(1/\sqrt{m})$ and $p \leqslant p_*$.

Corollary 5.3. *We can identify with probability at least $\Omega(1/m)$ any non-edge with the degrees of both ends no more than $2/p_*$ by a query on a p-random set, where $p = \Omega(1/\sqrt{m})$ and $p \leqslant p_*$.*

Lemma 5.4. *There are at most $1/p_*$ vertices that have degree more than $2/p_*$.*

Proof. Suppose that there are h vertices that have degree more than $2/p_*$. Let P be a p_*-random set. Given that one of the h vertices is included in P, the probability that P has no edge in G is at most $(1-p_*)^{2/p_*} \leqslant 1/e^2$. The probability that P contains none of the h vertices is at most $(1-p_*)^h$. Therefore, the probability P has no edge of G is at most

$$\left(1-(1-p_*)^h\right) \cdot \frac{1}{e^2} + (1-p_*)^h \cdot 1 \leqslant \frac{1}{e^2}\left(1+\left(e^2-1\right) \cdot e^{-p_* h}\right)$$

which should be no less than $1/2$. Thus we have $e^{-p_* h} \geqslant 1/e$. Therefore $h \leqslant 1/p_*$. □

Recalling that $p_* \geqslant 1/\sqrt{2m}$, we have the following.

Corollary 5.5. *There are at most $O(\sqrt{m})$ vertices that have degrees more than $2/p_*$.*

Algorithm 3 (*5-round algorithm*)

1. (*Estimate p_**) Let $p_i = 2^i/n$ for $i = 0, \ldots, \lfloor \log n \rfloor$. For each i, choose and query a p_i-random set $\Theta(\log n)$ times. Let the average outcome of edge-detecting queries on p_i-random sets be $1-\hat{N}_i$. Let $p' = (1/2)\min\{p_i \mid \hat{N}_i \leqslant 5/8\}$. Go to the 5th round if p' does not exist.
2. (*Low-degree edges*) Choose and query a p'-random set $\Theta((1/p'^2)\log n)$ times.
3. (*Degree estimation of high-degree vertices*) Let E' be the set of pseudo-edges after the second round. Let $G' = (V, E')$.
   (a) Divide V into two sets H and L according to their degrees in G'. L contains the vertices that have degrees at most $3/p'$ and H contains the rest of the vertices.
   (b) For each vertex u in H and each p_i, query $\Theta(\log n)$ times the union of $\{u\}$ and a p_i-random set.
   (c) Let $1-\hat{N}_i^u$ be the average outcome of random queries with probability p_i on vertex u. Let $p_u = \max\{p_i \mid \hat{N}_i^u \geqslant 1/5 + 1/(2e)\}$ if p_u exists.
4. (*Edges between high-degree and low-degree vertices*) For each vertex $u \in H$ such that p_u exists, query the union of $\{u\}$ and a p_u-random set $\Theta((1/p_u)\log n)$ times.
5. (*Cleanup*) Query every remaining pseudo-edge.

The 5-round algorithm is shown in Algorithm 3. Its correctness is guaranteed by the cleanup round, so our task is to bound the expected number of queries. For this analysis, we call a vertex a *low-degree vertex* if its degree is at most $2/p_*$ and call it a *high-degree* vertex otherwise. The non-edges consisting of two low-degree vertices are called *low-degree non-edges*. In the following, we will show that each round will succeed with probability at least $1-n^{-2}$ given that the previous rounds succeed.

First we show that with high probability p' exists and satisfies our requirement for the second round.

Lemma 5.6. *$p' \leqslant p_*$ and $p' = \Omega(1/\sqrt{m})$ with probability at least $1 - n^{-2}$.*

Proof. Let $p_j = \min\{p_i \mid p_i \geqslant p_*\}$. Obviously p_j exists and we have $p_j \leqslant 2p_*$. First we observe that with high probability

$$p' \leqslant \frac{1}{2} p_j \leqslant p_*.$$

The probability that the above inequality is violated is $\Pr[p' > (1/2)p_j] \leqslant \Pr[\hat{N}_j > 5/8]$. We know that $E[\hat{N}_j] = N(p_j) \leqslant N(p_*) = 1/2$. Let k be the number of times we query a p_j-random set. According to Hoeffding's inequality [8] (Proposition 3) (set $\epsilon = k(5/8 - 1/2) = k/8$),

$$\Pr[\hat{N}_j \geqslant 5/8] \leqslant e^{-k/32}.$$

We can make the probability at most $1/(2n^2)$ by asking $k = \Theta(\log n)$ queries.

Let $p_l = 2p'$. Since $\hat{N}_l \leqslant 5/8$ in our estimation step, by Hoeffding's inequality again,

$$\Pr\big[N(2p') > 3/4\big] = \Pr\big[N(p_l) > 3/4\big] = \Pr\big[E[\hat{N}_l] > 3/4\big] \leqslant 1/\big(2n^2\big)$$

when we set $k = \Theta(\log n)$. Therefore, we have $1 - m(2p')^2 \leqslant N(2p') \leqslant 3/4$, and hence $p' \geqslant 1/(4\sqrt{m})$. Thus with probability at least $1 - n^{-2}$, we have $1/(4\sqrt{m}) \leqslant p' \leqslant p_*$. □

Using Corollary 5.3, we can conclude that if the above inequalities are true, by asking $\Theta(m \log n)$ queries, we can guarantee with probability at least $1 - n^{-4}$ that a given low-degree non-edge is identified in the second round. So we can guarantee with probability at least $1 - n^{-2}$ that every low-degree non-edge is identified in the second round.

Suppose that we identify all of the low-degree non-edges in the second round. All the low-degree vertices must fall into L, since their degrees in G' are at most $3/p_*$ (which is at most $|H| \leqslant 1/p_*$ more than their true degrees).

However, L may also contain some high-degree vertices. At most $1/p_*$ high-degree vertices fall into L, and their degrees are bounded by $3/p'$. Note that both $1/p'$ and $1/p_*$ are $O(\sqrt{m})$. The total number of pseudo-edges incident with high-degree vertices in L is therefore bounded by $O(m)$. Also, the number of pseudo-edges between pairs of vertices in H is bounded by $O(m)$. As stated before, they can be identified in the cleanup round with $O(m)$ queries. We will therefore analyze only the behavior of non-edges between vertices in H and low-degree vertices in L in the third and fourth round.

We will now show p_u is what we want for each vertex $u \in H$. Let d_u denote the degree of vertex u.

Lemma 5.7. *For each $u \in H$, $1/(10d_u) \leqslant p_u \leqslant 1/d_u$ with probability at least $1 - n^{-3}$, given that the algorithm succeeds in the first and second rounds.*

Proof. Denote by $N^u(p)$ the probability that the union of $\{u\}$ and a p-random set has no edge. Using Hoeffding's inequality as in the proof of Lemma 5.6, by asking $\Theta(\log n)$ queries we can make $N^u(p_u) \geqslant 1/e$ true with probability at least $1 - (1/3)n^{-3}$. Note that $N^u(p_u) \leqslant (1 - p_u)^{d_u} \leqslant e^{-p_u d_u}$. Thus we can conclude that $p_u \leqslant 1/d_u$ is true with probability at least $1 - (1/3)n^{-3}$.

Assume $p_j^u = \max\{p_i \mid N^u(p_i) \geqslant 2/5\}$. First we observe that with high probability that $p_u \geqslant p_j^u$. The probability this inequality is violated is

$$\Pr\big[p_u \geqslant p_j^u\big] \geqslant \Pr\big[N^u(p_j) < 1/5 + 1/2e\big].$$

Again by Hoeffding's inequality, the probability can be made no more than $(1/3)n^{-3}$ by asking $\Theta(\log n)$ queries.

According to our choice of p_j^u, we have $N^u(p_{j+1}^u) < 2/5$. By Lemma 4.1 we know that

$$N^u\big(p_{j+1}^u\big) \geqslant \big(1 - 2p_j^u\big)^{d_u} \cdot N\big(2p_j^u\big).$$

As we just showed, $p_u \geqslant p_j^u$ is true with probability at least $1 - (1/3)n^{-3}$. Therefore, with probability at least $1 - (1/3)n^{-3}$

$$\frac{2}{5} > N^u\big(p_{j+1}^u\big) \geqslant (1 - 2p_u)^{d_u} \cdot N(2p_u) \geqslant (1 - 2p_u d_u) \cdot N(2p_u).$$

Since we already showed that $p_u \leqslant 1/d_u$ is true with probability at least $1 - (1/3)n^{-3}$ and we know that $\forall u \in H$, $d_u \geqslant 2/p_*$, we have

$$N(2p_u) \geqslant N\left(\frac{2}{d_u}\right) \geqslant N(p_*) \geqslant \frac{1}{2}$$

is true with probability at least $1 - (1/3)n^{-3}$. Thus we can conclude that $p_u \geqslant 1/(10d_u)$ with probability at least $1 - (2/3)n^{-3}$. □

In the third round, we can guarantee that $1/(10d_u) \leqslant p_u \leqslant 1/d_u$ is true for every $u \in H$ with probability at least $1 - n^{-2}$.

Let us assume the above inequality is true for every $u \in H$. Suppose v is a low-degree vertex and $\{u, v\}$ is a non-edge. Let P be a p_u-random set.

$$\Pr\{Q(P \cup \{u, v\}) = 0\} \geqslant (1 - p_u)^{d_u + 2/p_*} \cdot N(p_u).$$

Since $p_u \leqslant 1/d_u \leqslant p_*/2$, we have both $(1 - p_u)^{d_u+2/p_*} = \Omega(1)$ and $N(p_u) = \Omega(1)$. The probability that we choose v in one random query is p_u, which is $\Omega(1/d_u)$. Therefore, the probability $\{u, v\}$ is identified in one random query concerning u is $\Omega(1/d_u)$. By querying the union of $\{u\}$ and a p_u-random set $\Theta(d_u \log n)$ times, we can guarantee that $\{u, v\}$ is identified as a non-edge with probability at least $1 - n^{-4}$. Therefore, given that rounds one, two and three succeed, round four identifies every non-edge $\{u, v\}$ with $u \in H$ and v a low degree vertex, with probability at least $1 - n^{-2}$.

Given that the algorithm succeeds in rounds one through four, the only pseudo-edges that remain are either edges of G or non-edges between pairs of vertices in H or non-edges incident with the high degree vertices in L. As shown above, the total number of such non-edges is $O(m)$.

Finally, we bound the expected number of queries used by the algorithm. It is clear that in the event that each round succeeds, the first round uses $O(\log^2 n)$ queries; the second round uses $O(m \log n)$ queries; the third round uses $O(\sqrt{m} \log^2 n)$ queries; the fourth round uses $O(\sum_{u \in H} d_u \log n) = O(m \log n)$ queries; the fifth round uses $O(m)$ queries. The probability that each round fails is bounded by n^{-2}. The maximum number of queries used in case of failures is $O(n^2 \log n)$. Therefore in expectation the algorithm uses $O(m \log n + \sqrt{m} \log^2 n)$ queries. Note that this bound is $O(m \log n)$ if m is $\Omega(\log^2 n)$.

Therefore, we have the following theorem.

Theorem 5.8. *There is a Las Vegas 5-round algorithm that identifies any graph G drawn from the class of all graphs with n vertices and m edges using $O(m \log n + \sqrt{m} \log^2 n)$ edge-detecting queries in expectation.*

6. Hypergraph learning

In this section, we consider the problem of learning hypergraphs with edge-detecting queries. An edge-detecting query $Q_H(S)$ where H is a hypergraph is answered 1 or 0 indicating whether S contains all vertices of at least one hyperedge of H or not. The information-theoretic lower bound implies that any algorithm takes at least $\Omega(rm \log n)$ queries to learn hypergraphs of dimension r with m edges.

We show that no algorithm can learn hypergraphs of dimension r with m edges using $o((2m/r)^{r/2})$ queries if we allow the hypergraph to be non-uniform, even if we allow randomness. When m is large, say $\omega(r \log^2 n)$, this implies that there is no algorithm using only $O(r \log n)$ queries per edge when $r \geqslant 3$.

Theorem 6.1. *$\Omega((2m/r)^{r/2})$ edge-detecting queries are required to identify a hypergraph H drawn from the class of all hypergraphs of dimension r with n vertices and m edges.*

Proof. We generalize the lower bound argument from [1] for learning monotone DNF formulas using membership queries. Let r and k be integers greater than 1. Let $V_1, \ldots, V_r$ be pairwise disjoint sets containing k vertices each. For

$1 \leqslant i \leqslant r$ let $E_i = \{(u, v) \mid u, v \in V_i, u \neq v\}$. Thus, E_i is a clique of 2-edges on the vertices V_i. Consider a hypergraph H with vertices V including each V_i and edges

$$E = \bigcup_{i=1}^{r} E_i \cup \{v_1, v_2, \ldots, v_r\}$$

where $v_i \in V_i$ for $1 \leqslant i \leqslant r$. There are k^r such hypergraphs, one for each choice of an r-edge.

Even knowing the form of the hypergraph and the identity of the sets of vertices V_i, the learning algorithm must ask at least $k^r - 1$ queries if the adversary is adaptive. Every query that contains more than one vertex from some V_i is answered 1; therefore, only queries that contain exactly one vertex from each V_i yield any information about the r-edge characterizing H. An adversary may maintain a set $R \subseteq V_1 \times \cdots \times V_r$ consisting of the r-edges not queried so far. Each query with an r-edge may be answered 0 until $|R| = 1$, which means that the learning algorithm must make at least $k^r - 1$ queries to learn H. In terms of m, this is $\Omega((2m/r)^{r/2})$.

Even if the adversary is constrained to make a random choice of an r-edge T at the start of the algorithm and answer consistently with it, we show that $\Omega((2m/r)^{r/2})$ queries are necessary. Suppose $S_1, S_2, \ldots, S_q$ is the sequence of r-edges a randomized algorithm makes queries on. It is easy to see that $\Pr\{S_1 = T\} = 1/k^r$. And also we have $\Pr\{S_{i+1} = T \mid S_j \neq T,\ j \leqslant i\} \leqslant 1/(k^r - i)$ since each r-edge is equally likely to be T. Therefore, the probability that none of the S_i's equals T is at least $(k^r - q)/k^r$. When $q \leqslant k^r/2$, this is at least $1/2$. □

7. Open problems

We leave the following problems open. Reduce the number of queries needed for Algorithm 3 from $O(m \log n + \sqrt{m} \log^2 n)$ to $O(m \log n)$. Reduce the number of rounds of Algorithm 3 without substantially increasing the number of queries.

Acknowledgments

The conference version of this paper appears in COLT '04 [4]. We thank the anonymous referees for comments and corrections. This work was done while J. Chen was a PhD student in the Department of Computer Science of Yale University.

References

[1] D. Angluin, Queries and concept learning, Machine Learning 2 (1988) 319–342.
[2] N. Alon, V. Asodi, Learning a hidden subgraph, in: 31st International Colloquium on Automata, Languages and Programming, 2004, pp. 110–121.
[3] N. Alon, R. Beigel, S. Kasif, S. Rudich, B. Sudakov, Learning a hidden matching, in: The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002, pp. 197–206.
[4] D. Angluin, J. Chen, Learning a hidden graph using $O(\log n)$ queries per edge, in: The 17th Annual Conference on Learning Theory, Springer, 2004, pp. 210–223.
[5] D. Angluin, J. Chen, Learning a hidden hypergraph, J. Machine Learning Research 7 (2006) 2215–2236.
[6] R. Beigel, N. Alon, S. Kasif, M.S. Apaydin, L. Fortnow, An optimal procedure for gap closing in whole genome shotgun sequencing, in: RECOMB, 2001, pp. 22–30.
[7] V. Grebinski, G. Kucherov, Optimal query bounds for reconstructing a Hamiltonian cycle in complete graphs, in: Fifth Israel Symposium on the Theory of Computing Systems, 1997, pp. 166–173.
[8] W. Hoeffding, Probability inequalities for sums of bounded random variables, J. Amer. Statist. Assoc. 58 (1963) 13–30.